

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 04-177461

(43)Date of publication of application : 24.06.1992

(51)Int.Cl.

G06F 15/31
G06F 15/332

(21)Application number : 02-305035

(71)Applicant : MITSUBISHI ELECTRIC CORP

(22)Date of filing : 07.11.1990

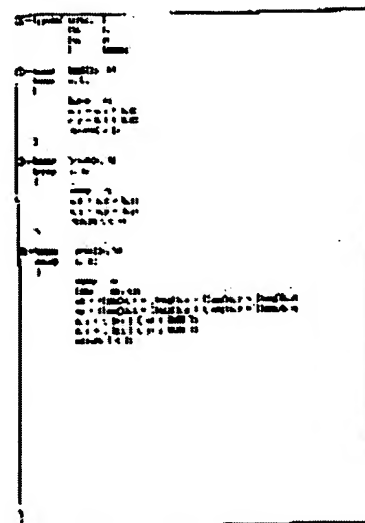
(72)Inventor : KEGASA MITSUYOSHI

(54) HIGH SPEED TECHNICAL CALCULATION METHOD

(57)Abstract:

PURPOSE: To perform a technical calculation at a high speed without using any expensive numerical arithmetic processor and at the same time to reduce the necessary capacity of a memory byperforming a numerical operation through an integer arithmetic and in a block floating point format.

CONSTITUTION: A fixed floating point arithmetic using an integer arithmetic is expressed in a block floating point number that showed the data in an exponent part or a coefficient part common to all data and a fixed decimal point of each data. Then a block floating point arithmetic applying an integer arithmetic is carried out. It is supposed that a decimal point is set at the 13th bit of an integer and then that 8192=2¹³ of an integer is equivalent to 1.01 of a fixed decimal point. In this fixed decimal point format, the value covering -4.0 (minimum) through about 3.9999 (maximum) can be expressed. The expressible number most approximate to 0 is about 0.000122 and a number approximate to 1.0 has the decimal accuracy equivalent to about 4 digits. The diagram shows a program used for calculation of a complex number and applying a fixed decimal point in a C language.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

⑫ 公開特許公報(A) 平4-177461

⑤ Int. Cl.⁵G 06 F 15/31
15/332

識別記号

Z 6798-5L
A 6798-5L

庁内整理番号

⑬ 公開 平成4年(1992)6月24日

審査請求 未請求 請求項の数 1 (全10頁)

⑭ 発明の名称 高速技術計算方法

⑯ 特 願 平2-305035

⑰ 出 願 平2(1990)11月7日

⑱ 発 明 者 毛 笠 光 容 兵庫県尼崎市塚口本町8丁目1番1号 三菱電機株式会社
通信機制作所内

⑲ 出 願 人 三菱電機株式会社 東京都千代田区丸の内2丁目2番3号

⑳ 代 理 人 弁理士 大岩 増雄 外2名

明 細 書

1. 発明の名称

高速技術計算方法

2. 特許請求の範囲

構造化された汎用の高級言語上で、複数個のデータを処理する技術計算において、整数演算を利用したブロック化浮動小数点演算もしくは固定小数点演算を用いたことを特徴とする、高速技術計算方法。

3. 発明の詳細な説明

〔産業上の利用分野〕

この発明は、コンピュータによる技術計算の高速化に関するものである。

〔従来の技術〕

第5図(a)、(b)は、CQ出版の「Cによる科学技術計算」小池慎一著の208ページ～211ページに掲載されていたC言語による複素数演算プログラムである。

(1)は複素数を表現する形式complexを定義しており、複素数の実数部を倍精度浮動小数点

(以降doubleと略す)のreであらわし、虚部をdoubleのimであらわしている。(2)は(1)で定義したcomplex型の加算用の関数caddであり、(3)はcomplex型の減算用の関数csul、(4)はcomplex型の乗算用の関数caulであり、(5)はcomplex型の除算用の関数cddivである。(6)はcomplex型の数の絶対値を求める関数cabsであり、(7)はdouble型の数2つを各成分とするcomplex型の数を求める関数to-complexである。

第6図は同じく「Cによる科学技術計算」の218ページに掲載されていた複素数演算を用いたFFTプログラムである。(10)はサブルーチン名cfftの宣言部、(11)はビットリバーサル操作部の先頭、(12)はバタフライ積部の先頭を示している。

第7図(a)、(b)は第6図を元にして図6のプログラムで省略されていたFFT演算後の割り算部を付加した実用的な複素数FFTプログラムである。(10)～(12)は第6図と同じもので

あり、(13)はFFT演算後の配列Zの要素をnで割る割り算部を示している。第8図は、ビットリバーサル部のフローチャート、第9図はバタフライ積部のフローチャート、第10図は割り算部のフローチャートである。

次に動作について説明する。

第5図の(2)の関数caddは複素数の変数a, bを引き数として受け取り、その和である複素数Zを返すもので、

Z.re(Zの実部)=a.re(aの実部)+b.re(bの実部)

Z.im(Zの虚部)=a.im(aの虚部)+b.im(bの虚部)

を計算している。

第5図の(3)の関数csubは同様に複素数の差を返すもので、X.re=a.re-b.re, X.im=a.im-b.imを計算している。

第5図の(4)の関数cmulは同様に複素数の積を返すもので、

X.re=a.re×b.re-a.im×b.im

X.im=a.re×b.im-a.im×b.re

を計算している。

第9図は前記プログラムのバタフライ演算部のフローチャートであり、第7図の(12)の部分の説明している。

第10図は第7図の(13)(第6図では省略されている)の割り算部のフローチャートである。

第7図のFFTプログラムcfftに複素数データの配列Z, FFTの次数を表わす整数1, FFTとRFTを選択する整数の変数f(f=1ならRFT, f=-1ならFFTが実行される)を引数として与えて実行させると、f=1なら配列Zのデータをフーリエ逆変換したデータが配列Zに求められる。f=-1なら、配列Zのデータをフーリエ変換したデータが配列Zに求められる。

FFTプログラムは、離散フーリエ変換を高速で行なうプログラムで時間的に変化する離散データg(k)の周波数スペクトルG(n)を、

$$G(n) = \frac{1}{N} \sum_{k=0}^{N-1} g(k) W_N^{nk}$$

という関係を用いて求めるプログラムである。但

第5図の(5)の関数cdivは同様に複素数の商を返すもので、

$$X.re = \frac{a.re \times b.re + a.im \times b.im}{b.re \times b.re + b.im \times b.im}$$

$$X.im = \frac{a.im \times b.re - a.re \times b.im}{b.re \times b.re + b.im \times b.im}$$

を計算している。又、分母が0の時はエラー表示をするようになっている。

第5図の(6)の関数cabsはaの絶対値をdoubleで返す関数で $|a| = \sqrt{(a.re)^2 + (a.im)^2}$ を計算している。

第5図の(7)の関数tocomplexはdoubleの変数xとyを引き数として受け取り、これを複素数Zに変換して返す関数でZ.re=x, Z.im=yを実行している。

第6図、第7図は複素数演算を用いたFFTのプログラムの例である。

第8図は前記のプログラムのビットリバーサル部のフローチャートであり、第7図の(11)の部分の説明している。

し、Nはデータ数で通常Zのベキ乗で表わされる数(2,4,8,...,128,256,512,1024,...)とし、 W_N は $e^{-j\frac{2\pi}{N}}$ で表わされる数である。

FFTプログラムは主に電気信号や機械的信号などがどのような周波数成分を持つかを求めるのに使われており、古くは大型コンピュータでしか使われていなかったが、近年ではパソコン上や測定器上でも使われるようになっている。

第5図の(2)～(7)の複素数演算を使用して複素数FFTを記述すると、第7図のように短いプログラムとなる。

第7図のプログラムをパソコンPC9801RX2上でボーランド社のCコンパイラTurbo-Cを用いて実行させると、データ数N=1024の時、数値演算プロセッサがない場合で約13秒必要であった。

数値演算プロセッサi80287-10を付けると、同じ演算が約3秒で実行できる。但しi80287-10は現状では4万円から6万円程度と高価である。

〔発明が解決しようとする課題〕

従来の技術計算方法は、以上のようにC言語で

サポートされている倍精度浮動小数点演算で行なわれていた。倍精度浮動小数点演算では各データを64ビットの指数方式で表現している為、10進数では15桁もの高精度が得られるが、その反面、加減乗除等の基本的な演算でも整数演算の100~1000倍の時間を要し1データで8バイトのメモリを要する。この為、従来の技術計算プログラムでは長時間を要しかつ多くのメモリを必要とした。浮動小数点演算を高速化する為、数値演算プロセッサが存在するが、かなり高価である為、汎用のパーソナルコンピュータ等では実装されていないものが多い。

この発明は、上記のような問題点を解消するためになされたもので、高価な数値演算プロセッサを使わずに、短時間で結果を求めることができ、かつ必要なメモリの容量も少ない高速技術計算プログラムを得ることを目的とする。

〔課題を解決するための手段〕

この発明に係る高速技術計算方法は、データを固定小数点またはブロック浮動小数点で表わし、

整数演算を利用した固定小数点演算もしくは、ブロック浮動小数点で計算を行なうものである。

〔作用〕

この発明における高速技術計算方法は、必要以上に高精度であり多くのメモリを必要とし、演算に長い時間を要する倍精度浮動小数点演算のかわりに整数演算を利用した固定小数点演算又は、データを全データ共通の指数部又は係数部と各データごとの固定小数点で表わしたブロック浮動小数点で表わし、整数演算を利用したブロックの浮動小数点演算で行なうことで技術計算を高速化しかつデータを格納するのに必要なメモリー容量を削減している。

〔実施例〕

以下この発明の一実施例を説明する。

本実施例では、固定小数点を整数を使って表現している。整数はコンピュータによって表現が異なるが、例えばパーソナルコンピュータPC9801シリーズ用のCコンパイラTurbo-C ver2.0(ポーランド社)では、整数は16ビットの符号付き整数

型(以降intと略す)を標準としている。16ビットの符号付き整数では 2^{15} 通りの数が表現でき、最小-32768~最大32767が表現できる。

本実施例ではこの整数の13ビット目に小数点があるものと仮定して整数の8192= 2^{13} が固定小数点の1.01に相当するものと設定した。この固定小数点フォーマットでは、最小-4.0~最大約3.9999が表現できる。表現できる0に最も近い数は約0.000122であり1.0付近の数なら10進で約4桁分の精度がある。

この固定小数点での演算は、加減算なら整数と全く同じ演算で実行できるが、乗算では $1.0 \times 1.0 = 1.0$ となる様に調整する必要がある。又、16ビットの整数の積は、最大約10億となり16ビット符号付整数では表現できなくなる。そこで乗算では、各整数を32ビット符号付整数型(以降long intと略す)に直して計算する必要がある。32ビット符号付整数では最小約-2億~最大約2億までの数が表現できる。long intで乗算後8192で割り、int型に戻してやれば乗算ができ

る。例えば、固定小数点数 $a = 2.0, b = 0.5$ の積を固定小数点のCに代入する演算をC言語で表記すると、

```
int a,b,c;
```

```
a=(int)(2.0*8192.0);
```

```
b=(int)(0.5*8192.0);
```

```
c=(int)((long int)a*(long int)b/8192);
```

となる。但し(int)はintへの型変換を指示するキャスト演算子である。又(long int)はlong intへの型変換を指示するキャスト演算子である。固定小数点の割り算も同様にlong intで計算する必要がある。例えば $a = 1.0, b = 2.0$ でa割るbをcに代入するには

```
int a,b,c;
```

```
a=(int)(1.0*8192.0);b=(int)(2.0*8192.0);
```

```
c=(int)((long int)a/8192/(long int)b);
```

とC言語で記述できる。

このように固定小数点の演算を整数演算の組み合わせで実行することが可能である。

さらに複素数を2つの固定小数点数で表現する

ことも可能である。つまり複素数の実部を1つの固定小数点数で、虚部をもう1つの固定小数点数で表現することができる。これをC言語でbcomp型と定義すると以下になる。

```
typedef struct{int x; int y;}bcomp;
bcomp Z1,Z2,Z3;
```

この場合複素数Z1の実部がZ1.xに、虚部がZ1.yに対応する。Z1=0.7+j0.7, Z2=0.5+j0.8の和をZ3に代入するには上の定義式にて宣言した後

```
Z1.x=(int)(0.7*8192.0);
Z1.y=(int)(0.7*8192.0);
Z2.x=(int)(0.5*8192.0);
Z2.y=(int)(0.8*8192.0);
Z3.x=Z1.x+Z2.x;
Z3.y=Z1.y+Z2.y;
```

と書けば良い。ここで、わかるように浮動小数点型数から固定小数点数を利用した複素数への変換や、複素数の計算をそのまま記述すると大変見にくくかつ書くのも大変であるので、C言語の関数

割れば良いのであるが、分子は2つの固定小数点数の積の和であり、例えば(2.0+j0.0)/(2.0+j0.0)を計算する場合分子の実部は整数で書けば(16384)²+0=268435456であり、これに8192を乗ずると約2.2×10¹²となりlong intでは表現できなくなる。long intの最大値は約2×10⁹であり、先の例の268435456の8倍であるので商の計算ではたとえば、被除数及び除数の絶対値が2以下という条件付きで、分子は8倍までが可能である。そこで、分子8倍して分母は8192÷8=1024で割ってから割り算を実行し、その後long intからint型へ戻すように変更した。この為、除数の絶対値が整数で√1024=32固定小数点で0.004より小さいとエラーとなる。この時は、エラー表示をするようにした。

除算をさらに高速化するには、分子への積をやめて分母を8192で割れば良いが、この場合は除数の絶対値が整数で√8192≈90固定小数点で0.01より小さいとエラーとなる。

としてやるのが好ましい。

第1図(a),(b)はC言語で固定小数点を利用した複素数計算用のプログラムである。最初の宣言部は上記説明のものと全く同じである。

(2)の関数bcaddは加算用の関数であり、従来の実施例の関数caddのcomplexの部分をbcompに変えたものである。

(3)の関数bcsubは減算用の関数であり、これも従来の実施例の関数csubのcomplexの部分をbcompに変えたものである。

(4)の関数bcmulは積を求める関数であり、従来の実施例の関数cmulのcomplexの部分をbcompに変え、さらに実部、虚部各部の乗算を求める左辺で整数の型をintからlong intに変換してから乗算を行ない固定小数点に戻す為、8192で割ってから型をint型に戻すように書き変えている。

(5)の関数bcddivは商を求める関数であり、従来の実施例の関数cddivのcomplexをbcompに変え、整数の型変換を行なっている。固定小数点の商であるので被除数へ8192を乗じてから、除数で

除算の精度を上げるには、たとえば被除数及び除数の絶対値を1以下という条件にしてやり、2²¹+(8192)²=32の余裕にして分子を32倍し、分母を256で割ってから割り算を行なえば良い。この場合、除数の絶対値が整数で√256=16、固定小数点で0.002より小さい時エラーとなる。

第1図において(6)は、複素数の絶対値を求める関数bcabsである。従来の実施例のcabsのcomplexの部分をbcompにおき変えたものである。

第2図において(7)は、絶対値が2以下のdoubleの数2つをそれぞれ実部、虚部とするbcompの数を求める関数tobcompであり、従来の実施例のtocomplexとほぼ同じであるが、固定小数点フォーマットに直す為に8192を掛けている。

(8)は複素数bcompの数の配列Z[]のn個の要素を2で割るサブルーチンbcnormであり、後で述べるFFTプログラムで使用している。

(9)は同じくbcompの配列Z[]のn個1要素をその各成分(実部、虚部)の絶対値の最大が1以下になるように調整するサブルーチン

bcnorm2である。絶対値の最大値を1以下にする為にZ〔 〕の全要素を変数constで割り、配列Z〔 〕の共通の指数部に相当する変数ampをconst倍している。このサブルーチンでは、Zとampよりなるブロック浮動小数点フォーマットをサポートしている。

(10)は、上で述べた複素演算を利用した複素FFTプログラムである。従来の実施例とほぼ同じアルゴリズムであるが、従来の実施例ではデータ数Nでの割り算をプログラムの最後でまとめて行なっていたが、本プログラムではバタフライ積演算を1組行なう度にデータをサブルーチンbcnormを使って2で割ることにより、Nでの割り算を実行している。FFTプログラムでは、バタフライ積を1組行なう度にデータの最大値が約2倍に増える性質があるので固定小数点途中でオーバーフローすることがないように毎回2で割ることにより、Nでの割り算を実行させている。

この複素FFTプログラムのアルゴリズムの異なる部分を説明しているフローチャートを第11

図、第12図に示す。第11図において(13)は追加されたbcnormサブルーチンの実行位置を示している。第9図の従来のプログラムで⑧で示すNでの割り算プログラムへ移行するかわりに、第11図ではサブルーチンを終了してメインプログラムへ戻っている。

図12はbcnormサブルーチンのアルゴリズムを示すフローチャートで、配列Zの各要素を2で割っている。この様に固定小数点フォーマットの複素数bcmpを使って加減乗除算や、複素数FFTが実現できる。

第4図は、上記の複素数FFTプログラムをテストする為に作られたC言語のメインプログラムであり、1024個のデータの実部がサインカーブを描くよう設定し、虚部は全て0になるよう設定し、複素数FFTプログラムを実行している。

つまり、時間軸で正弦振動をしているようなデータを入力した時の周波数スペクトラムを求めている。

pr_timeというサブルーチンは時刻を表示する

プログラムでこのFFTプログラムで消費した時間を求めさせている。

このメインプログラムを実行させて時間を測った結果1024点のFFTが約2秒で完了した。

従来の実施例に比べて約6倍の高速化が実現できた。数値演算プロセッサを使用したものよりも少し速い。但し精度は2桁～3桁程度である。例えば、8ビットのA/Dコンバータ等で入力されたデータの処理用としてはこの精度は十分な値であると思われる。又結果を画面(400ドット×640ドット)にグラフィック表示するようなアプリケーションプログラムでも、これだけの精度があれば十分である。

なお、上記実施例ではC言語上でプログラムを記述したがPASCAL等の他の構造化した高級言語でも同様のプログラムが作成できる。

又、上記実施例ではFFTを実行するプログラムを記述したが、同じような複数個のデータを処理する技術計算プログラムなら同様の手法で高速化が可能である。

又、上記実施例ではFFTを複素数演算を用いて計算しているが、複素数の実数部と虚数部のデータを分けてそれぞれ別の配列にして、通常の実数演算を用いて計算することも可能である。

又、上記実施例で使用したデータの複素数表現、複素数演算を用いて各種の技術計算を行なうことも可能であるが、技術計算の種類によっては固定小数点の精度が10進数で約4桁である為、演算中に大きな誤差を生じる場合があるので注意が必要である。

〔発明の効果〕

以上のようにこの発明によれば、数値演算を整数演算を利用してブロック浮動小数点フォーマットで行なったので、高価な数値演算プロセッサを使うことなく、高速で技術計算を行なうことができる。かつ必要とするメモリ容量も削減することができる。

4. 図面の簡単な説明

第1図はこの発明の一実施例による複素数の加減乗除と絶対値を求める関数の方法を示す図、第

2図はこの発明の一実施例による複素数の処理方法を示す図、第3図はこの発明の一実施例による複素FFTを示す図、第4図はこの発明の一実施例による図、第5図は従来の複素数演算用関数の図、第6図は従来の複素数演算を用いたFFT図、第7図は従来の実用的な複素数FFT図、第8図は従来及びこの発明の一実施例共通の複素数FFTプログラムのビットリバーサル部のフローチャート、第9図は従来の複素数FFTプログラムのバタフライ積部のフローチャート、第10図は従来の複素数FFTプログラムの割り算部のフローチャート、第11図はこの発明の一実施例による複素数FFTプログラムのバタフライ積部のフローチャート、第12図はこの発明の一実施例による複素数FFTプログラムのサブルーチンBCNORMのフローチャートである。

(1)は複素数の構造体宣言、(2)は複素数の加算用関数、(3)は複素数の減算用関数、(4)は複素数の乗算用関数、(5)は複素数の除算用関数、(6)は複素数の絶対値を求める関

数、(7)は実数2つから複素数を作る関数、(8)は複素数の配列のn個の要素を2で割るサブルーチン、(9)は複素数の配列の要素の各成分の最大値を1より小さくするサブルーチン、(10)は複素数FFTサブルーチン、(11)はビットリバーサル部、(12)はバタフライ積部、(13)は割り算部を示す。

なお、図中、同一符号は同一もしくは相当部分を示す。

代理人 大 岩 増 雄

```

1)-typedef struct {
    int x;
    int y;
} bcomp;

2)-bcomp badd(a, b)
bcomp c;
{
    c.x = a.x + b.x;
    c.y = a.y + b.y;
    return (c);
}

3)-bcomp bsub(a, b)
bcomp c;
{
    c.x = a.x - b.x;
    c.y = a.y - b.y;
    return (c);
}

4)-bcomp bmul(a, b)
bcomp c;
{
    long tx, ty;
    tx = (long)a.x * (long)b.x - (long)a.y * (long)b.y;
    ty = (long)a.x * (long)b.y + (long)a.y * (long)b.x;
    c.x = (int) (tx / 8192);
    c.y = (int) (ty / 8192);
    return (c);
}

```

図 1 (a)

```

5)-bcomp bdiv(a, b)
bcomp c;
{
    long tx, ty;
    tx = (long)a.x * (long)b.x + (long)a.y * (long)b.y;
    ty = (long)a.x * (long)b.y - (long)a.y * (long)b.x;
    if (tx == 0) fprintf(stderr, "Divide by 0 error in bdiv\n");
    tx = (long)a.x * (long)b.x + (long)a.y * (long)b.y;
    ty = (long)a.x * (long)b.y - (long)a.y * (long)b.x;
    c.x = (int) (tx / 8192);
    c.y = (int) (ty / 8192);
    return (c);
}

6)-double bcabs(a)
double x, y;
{
    x = (double)a.x;
    y = (double)a.y;
    x = sqrt(x * x + y * y) / 8192.0;
    return (x);
}

```

図 1 (b)

```

7) - bcomp tobcomp(a, b)
double
{
    bcomp c;
    c.x = (int) ( a * 8192.0 );
    c.y = (int) ( b * 8192.0 );
    return ( c );
}

```

```

8) - void bcompa(s, a)
bcomp
{
    int i;
    for( i = 0; i < a; i++)
    {
        (s).x /= 2;
        (s).y /= 2;
        s++;
    }
}

```

図 2 (a)

```

9) - void bcompa2(s, a, asp)
bcomp
{
    int maximum, i, const;
    bcomp sp;
    p = 1;
    maximum = 0;
    for ( i = 0; i < a; i++)
    {
        if ( abs((sp).x) > maximum) maximum = abs((sp).x);
        if ( abs((sp).y) > maximum) maximum = abs((sp).y);
    }
    const = 1;
    while ( maximum > 8192)
    {
        maximum /= 2;
        asp += 2;
        const += 2;
    }
    if ( const == 1 ) return;
    p = 1;
    for ( i = 0; i < a; i++)
    {
        (sp).x /= const;
        (sp).y /= const;
    }
}

```

図 2 (b)

```

10) - void cff(x, l, f)
bcomp
{
    int i, f;
    int s, av2, ip, l, j, h, u, lo, lei;
    bcomp t, w, u;
    double c, s;
    s = (int) pow(2.0, (double) l);
    av2 = s / 2;
    for (i=0; i<s; i++)
    {
        if ( i<av2 )
        {
            t = x[i];
            z[i] = x[j];
            z[j] = t;
        }
        h = av2;
        while ( h<s )
        {
            j = h;
            h /= 2;
            j += h;
        }
    }
    for ( u=1; u<=l; u++)
    {
        lo = (int) pow(2.0, (double) u);
        lei = lo / 2;
        u = tobcomp(1.0, 0.0);
        c = cos( 8.0 * PI / (double) lei );
        s = ( (double) f ) * sin( 8.0 * PI / (double) lei );
        w = tobcomp( c, s );
        for ( j=0; j<lei/2; j++)
        {
            for ( i=j; i < lei; i += lei )
            {
                ip = i + lei;
                t = bcomp( z[ip], u );
                z[ip] = bcomp( z[i], t );
                z[i] = bcomp( z[i], t );
            }
            u = bcomp( u, w );
        }
    }
    bcompa( s, a );
}

```

図 3

```

void main(void)
{
    double ds, x, y;
    int i, f, f;
    bcomp z[1024];
    ds = 2.0 * 8.0 * PI / 1024.0;
    z = 0;
    for( i=0; i < 1024; i++)
    {
        z[i] = tobcomp( sin(x), 0.0 );
        x += ds;
    }
    printf(" Time domain Va");
    for( i=0; i < 8; i++)
    {
        x = ( (double) z[i].x ) / 8192.0;
        printf( " z[%2d].x = %1f Va", i, x );
    }
    pr_time();
    cff( z, 10, -1 );
    pr_time();
    printf(" Frequency domain Va");
    for( i=0; i < 8; i++)
    {
        x = ( (double) z[i].x ) / 8192.0;
        y = ( (double) z[i].y ) / 8192.0;
        printf( " z[%2d].x = %1f, z[%2d].y = %1f Va", i, x, i, y );
    }
}

```

図 4


```

① - typedef struct complx
{
    double re;
    double ie;
} complx;

complx cadd(), csub(), cmul(), cdiv(), conj(), cexp(), cpow(),
crt(), locomplex(), cisp(),
double cabs(), carg();

② - complx cadd(a, b)
complx a, b;
{
    complx s;

    s.re = a.re + b.re; s.ie = a.ie + b.ie;
    return s;
}

③ - complx csub(a, b)
complx a, b;
{
    complx s;

    s.re = a.re - b.re; s.ie = a.ie - b.ie;
    return s;
}

④ - complx cmul(a, b)
complx a, b;
{
    complx s;

    s.re = a.re * b.re - a.ie * b.ie;
    s.ie = a.re * b.ie + a.ie * b.re;
    return s;
}

```

図 5 図 (a)

```

⑤ - complx cdiv(a, b)
complx a, b;
{
    complx s;
    double d;

    d = b.re + b.ie * b.ie;
    if (d == 0)
    {
        printf("\n?divided by zero error!!!\n");
        d = 1.0;
    }
    s.re = (a.re * b.re + a.ie * b.ie) / d;
    s.ie = (a.ie * b.re - a.re * b.ie) / d;
    return s;
}

⑥ - double cabs(a)
complx a;
{
    return (sqrt(a.re * a.re + a.ie * a.ie));
}

double carg(a)
complx a;
{
    if ((a.re == 0) && (a.ie == 0))
    {
        printf("\n?arg zero error!!!\n");
        return 0.0;
    }
    else return (atan2(a.ie, a.re));
}

complx conj(a)
complx a;
{
    complx s;

    s.re = a.re; s.ie = -a.ie;
    return s;
}

⑦ - complx locomplex(x, y)
double x, y;
{
    complx s;

    s.re = x; s.ie = y;
    return s;
}

```

図 5 図 (b)

```

⑧ - int cfft(x, l, f) /* FFT by using complex type */
complx *x; /* adapted by S. Itoke from Ref(3) */
double f; /* f = 1.0... fft */
int l; /* 1.0... rft */
{
    int a, n2, ip, i, j, k, m, le, lcl;
    complx t, u, w;
    double s, c;

    n = ipow2(l); n2 = n / 2;

    /* bit reversal operation */
    for (i = 0; i < n; i++)
    {
        if (i < j)
        {
            t = x[i]; x[i] = x[j]; x[j] = t;
        }
        k = n2;
        while (k < i)
        {
            j = k; k = i; i = k;
        }
        j += k;
    }

    /* main loop */
    for (a = 1; a < n; a++)
    {
        le = ipow2(a); lcl = le / 2;
        w = locomplex(1.0, 0.0);
        c = cos(PI / (double)le);
        s = f * sin(PI / (double)le);
        u = locomplex(c, s);
        for (j = 0; j < le / 2; j++)
        {
            for (i = j; i < n; i += le)
            {
                ip = i + lcl;
                t = cmul(x[ip], u); x[ip] = csub(x[i], t);
                x[i] = cadd(x[i], t);
            }
            a = conj(a, s);
        }
    }
}

```

図 6 図

```

#include <stdio.h>
#include <math.h>
#include <complex.h>

⑨ - void cfft(x, l, f)
complx *x;
int l, f;
{
    int a, n2, ip, i, j, k, m, le, lcl;
    complx t, u, w;
    double s, c;

    n = (int)pow(2.0, (double)l);
    n2 = n / 2;

    /* bit reversal operation */
    for (i = 0; i < n; i++)
    {
        if (i < j)
        {
            t = x[i];
            x[i] = x[j];
            x[j] = t;
        }
        k = n2;
        while (k < i)
        {
            j = k; k = i; i = k;
        }
        j += k;
    }
}

```

図 7 図 (a)

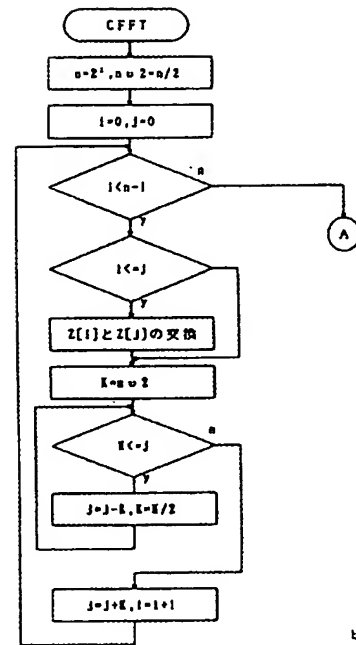
```

10- for ( a=1; a<=1; a++)
{
    le = (int)pow(2.0,(double)a);
    let = le / 2;
    u = tocomplex(1.0, 0.0);
    c = cos( 8 * PI / (double) let );
    s = ( (double) c ) * sin( 8 * PI / (double) let );
    u = tocomplex( c, s );
    for ( j=0; j<le/2; j++)
    {
        for ( i=j; i<a; i+=le )
        {
            ip = i + let;
            t = cexp( z[ip], u );
            z[ip] = cexp( z[i], t );
            z[i] = cexp( z[i], t );
        }
        a = cexp( u, u );
    }
}

11- if ( f < 0 ) for ( i = 0; i < a; i++)
{
    z[i].x /= (double) a;
    z[i].y /= (double) a;
}

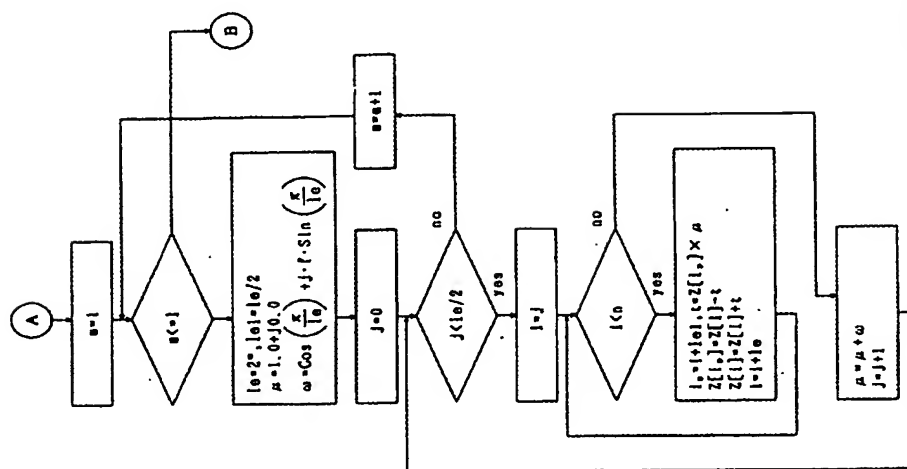
```

図 7 (c)



ビットリバーサル部

図 8



バタフライ部

図 9

図10

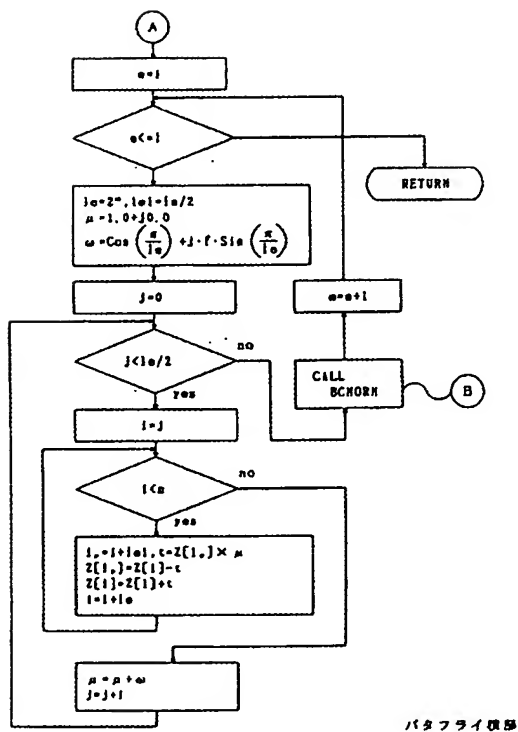
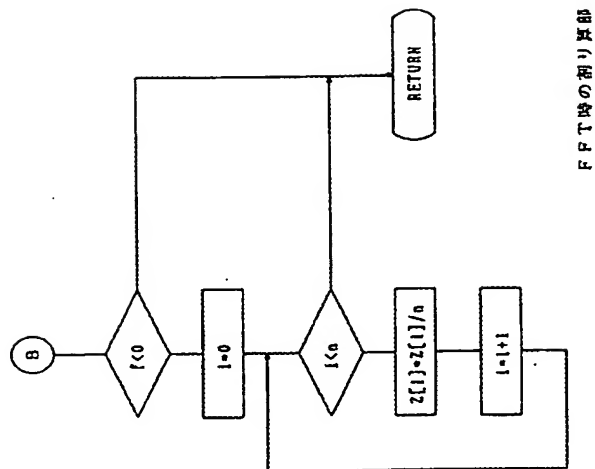


図11

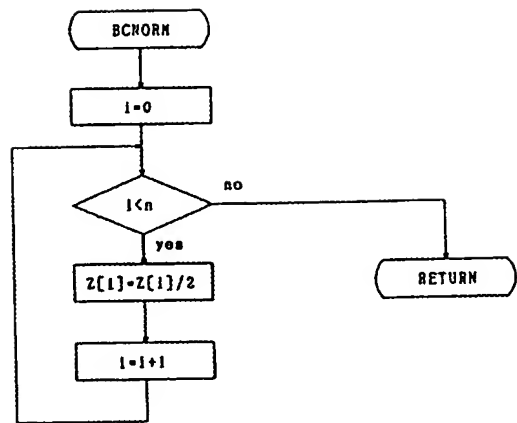


図12